

Algorithmen in Zellularautomaten

3. Endliche Muster und Konfigurationen

Thomas Worsch

Fakultät für Informatik
Institut für Theoretische Informatik

Sommersemester 2018

Überblick

- Weitere wichtige Begriffe
- Simulationen zwischen ZA und TM
- Erkennung formaler Sprachen mit ZA

Überblick

- Weitere wichtige Begriffe
- Simulationen zwischen ZA und TM
- Erkennung formaler Sprachen mit ZA

Definition (Ruhezustände und tote Zustände)

- ▶ Eine Teilmenge $P \subseteq Q$ heißt *Ruhemenge* oder *passiv*, wenn für alle $\ell : N \rightarrow Q$ gilt:
wenn für alle $n \in N$: $\ell(n) \in P$ ist, dann $\delta(\ell) = \ell(0)$.
- ▶ Ein Zustand $q \in Q$ heißt genau dann *Ruhezustand*, wenn $\{q\}$ Ruhemenge ist (also falls $\delta(q, q, \dots, q) = q$ ist).
- ▶ Ein Zustand $t \in Q$ heißt genau dann *tot*, wenn für alle $\ell : N \rightarrow Q$ mit $\ell(0) = t$ gilt: $\delta(\ell) = t$.
- ▶ Jeder Zellularautomat besitze ausgezeichneten Ruhezustand q .

Beispiele

- ▶ WIREWORLD: $\{\blacksquare, \square\}$ ist Ruhemenge und \blacksquare ist tot.
- ▶ Jeder tote Zustand ist Ruhezustand.
- ▶ Jeder Zustand einer Ruhemenge ist Ruhezustand.
- ▶ Ein ZA kann Ruhezustände q_1, q_2 besitzen, die zusammen *keine* Ruhemenge bilden.

Erinnerung (Eingaben für TM)

- ▶ Eingabealphabet $A \subseteq B$
- ▶ Anfangszustand s_0
- ▶ Blanksymbol \square
- ▶ Die *Anfangskonfiguration* zu Eingabe $w = w_1 \cdots w_n \in A^n$ ist $c_w = (s_0, b_w, 1)$ mit

$$b_w(i) = \begin{cases} w_i & \text{falls } 1 \leq i \leq n \\ \square & \text{sonst} \end{cases}$$

Definition (Muster)

für Zellularautomat C mit $R = \mathbb{Z}^d$ und Ruhezustand q :

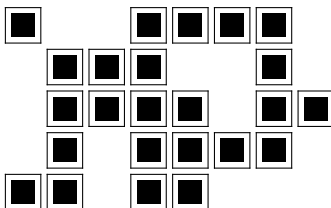
- ▶ (d -dimensionales) *Muster* für C : Abbildung $m : T \rightarrow Q \setminus \{q\}$,
 $T \subset R$: der *Träger* von m
- ▶ Zu Muster m gehörende *Musterkonfiguration* $c_m : R \rightarrow Q$:

$$c_m(i) = \begin{cases} m(i) & \text{falls } i \in T \\ q & \text{sonst} \end{cases}$$

- ▶ *Muster* $m : T \rightarrow Q \setminus \{q\}$ kommt in Konfiguration c an der Stelle j vor, wenn für alle $i \in T$ gilt: $c(j + i) = m(i)$.

Beispiele von Mustern

- ▶ eindimensional: Wörter $w \in Q^+$ kann man als Muster $w : \{1, \dots, |w|\} \rightarrow Q$ auffassen und umgekehrt.
- ▶ zweidimensional: Rechtecke $m : \mathbb{N}_n \times \mathbb{N}_m \rightarrow Q$.
- ▶ „unförmige“ zweidimensionale Muster:



- ▶ höherdimensionale Muster ...

Definition

- ▶ *Eingabe* eines Musters m in einen ZA:
zu m gehörende Musterkonfiguration als *Anfangskonfiguration*
- ▶ Konfiguration c ist genau dann *Endkonfiguration*, wenn $\Delta(c) = c$.

Aber was ist das *Ergebnis einer Berechnung*?

- ▶ je nach Aufgabenstellung passend definiert

Definition

- ▶ Für Ruhezustand q und Konfiguration $c : R \rightarrow Q$ ist $T(c) = \{i \mid c_i \neq q\}$ der *Träger* von c .
- ▶ Eine Konfiguration heißt *endlich*, wenn ihr Träger endlich ist.
(manche finden, diese Redeweise geht zu weit)

Beachte

Im Laufe einer Berechnung entstehen aus endlichen Konfigurationen wieder endliche Konfigurationen.

Überblick

- Weitere wichtige Begriffe
- Simulationen zwischen ZA und TM
- Erkennung formaler Sprachen mit ZA

Satz (Simulation von TM durch ZA und umgekehrt)

1. Zu jeder Ein-Kopf-TM T gibt es einen eindimensionalen ZA C und eine Codierungsfunktion cod , so dass für jede TM-Konfiguration c gilt:

$$\Delta_C(\text{cod}(c)) = \text{cod}(\Delta_T(c))$$

2. Zu jedem eindimensionalen ZA C gibt es eine TM T und eine Codierungsfunktion cod , so dass für jede Konfiguration c mit endlichem Träger (!) ein $t \in \mathbb{N}$ existiert mit:

$$\Delta_T^t(\text{cod}(c)) = \text{cod}(\Delta_C(c))$$

3. In beiden Fällen ist cod injektiv und „sehr einfach“.

Beweisskizze zu 2.

- ▶ Wie könnte man das machen?
- ▶ Details an der Tafel

Aufgabe

- ▶ Wie geht eine Simulation, die nicht doppelten Speicher benötigt?

Aufgabe

- ▶ gegeben ZA (R, Q, N, δ)
- ▶ $M \subseteq R$ Teilmenge von Zellen
- ▶ $c : R \rightarrow Q$ eine Konfiguration mit

$$\forall i \in M : c_i = \Delta(c)_i$$

- ▶ betrachte Konfigurationsübergang $\Delta(c) \mapsto \Delta^2(c)$:
Was kann man sagen?

Überblick

- Weitere wichtige Begriffe
- Simulationen zwischen ZA und TM
- Erkennung formaler Sprachen mit ZA

Definition (Erkennung formaler Sprachen)

- ▶ *Eingabealphabet* $A \subset Q \setminus \{q\}$
- ▶ Zu $w = w_1 \cdots w_n \in A^+$ gehörige *Anfangskonfiguration* c_w ist die zu $m : \{(1, 0, \dots, 0), \dots, (|w|, 0, \dots, 0)\} \rightarrow A : (i, 0, \dots, 0) \mapsto w_i$ gehörende Musterkonfiguration.
- ▶ Menge *akzeptierender Endzustände* $F_+ \subset Q \setminus A$
- ▶ Endkonfiguration c ist *akzeptierend*, falls $c((1, 0, \dots, 0)) \in F_+$, sonst *ablehnend*.

Definition (Erkennung formaler Sprachen 2)

- ▶ c^0, c^1, \dots, c^k ist *Berechnung*, falls c^k die einzige vorkommende Endkonfiguration ist und für alle $t \in \mathbb{N}$ mit $0 \leq t < k$ gilt: $c^{t+1} = \Delta(c^t)$.
- ▶ *Von einem Zellularautomaten erkannte Sprache* $L \subseteq A^+$:

$$L = \left\{ w \mid \begin{array}{l} \text{es existiert akzept. Endkonf. } c_f \\ \text{und } k \in \mathbb{N}_+ \text{ mit } \Delta^k(c_w) = c_f \end{array} \right\}$$

Erstes Beispiel: Erkennung einer regulären Sprache

Aufgabe: $A = \{a, b\}$

akzeptiere $L = \{a\}^+$

Erstes Beispiel: Erkennung einer regulären Sprache

Aufgabe: $A = \{a, b\}$

akzeptiere $L = \{a\}^+$

Problem: Prüfe, ob in der Eingabe ein b vorkommt!

Erstes Beispiel: Erkennung einer regulären Sprache

Aufgabe: $A = \{a, b\}$
akzeptiere $L = \{a\}^+$

Problem: Prüfe, ob in der Eingabe ein b vorkommt!

Idee: Simuliere den endlichen Automaten,
der einmal (von hinten nach vorne) über die Eingabe geht.

Erstes Beispiel: Erkennung einer regulären Sprache

Aufgabe: $A = \{a, b\}$
akzeptiere $L = \{a\}^+$

Problem: Prüfe, ob in der Eingabe ein b vorkommt!

Idee: Simuliere den endlichen Automaten,
der einmal (von hinten nach vorne) über die Eingabe geht.

Wähle: $N = \{0, 1\}$ und $Q = \{ a , b , \sqcup , <+ , <- \}$.

Beispielrechnung für eine „gute“ Eingabe

...	┌	a	a	a	a	a	└	...
...	┌	a	a	a	a	<+	└	...
...	┌	a	a	a	<+	<+	└	...
...	┌	a	a	<+	<+	<+	└	...
...	┌	a	<+	<+	<+	<+	└	...
...	┌	<+	<+	<+	<+	<+	└	...

Beispielrechnung für eine „schlechte“ Eingabe

...	┌	a	b	a	a	a	└	...
...	┌	a	b	a	a	<+	└	...
...	┌	a	b	a	<+	<+	└	...
...	┌	a	b	<+	<+	<+	└	...
...	┌	a	<-	<+	<+	<+	└	...
...	┌	<-	<-	<+	<+	<+	└	...

Lokale Überföhrungsfunktion

$\ell(0)$	$\ell(1)$	$\delta(\ell)$
a	┐	<+
b	┐	<-
a	<+	<+
b	<+	<-
a	<-	<-
b	<-	<-

In allen anderen Fällén: $\delta(\ell) = \ell(0)$.

Aufgabe

Konstruiere (möglichst präzise) einen ZA, der

$$L = \{a^m b^m \mid m \in \mathbb{N}_+\} \subset \{a, b\}^+$$

akzeptiert.

Zweites Beispiel: Erkennung einer Palindromsprache

Aufgabe: $A = \{a, b\}$

akzeptiere $L = L_{\text{pal}} = \{v xv^R \mid v \in A^* \wedge x \in A\}$

Zweites Beispiel: Erkennung einer Palindromsprache

Aufgabe: $A = \{a, b\}$

akzeptiere $L = L_{\text{pal}} = \{v xv^R \mid v \in A^* \wedge x \in A\}$

Problem: Prüfe, ob

1. erstes und letztes Symbol übereinstimmen und ob
2. zweites und vorletztes Symbol übereinstimmen und ob
3. drittes und drittletztes Symbol übereinstimmen und
4. usw. ...

Zweites Beispiel: Erkennung einer Palindromsprache

Aufgabe: $A = \{a, b\}$

akzeptiere $L = L_{\text{pal}} = \{v xv^R \mid v \in A^* \wedge x \in A\}$

Problem: Prüfe, ob

1. erstes und letztes Symbol übereinstimmen und ob
2. zweites und vorletztes Symbol übereinstimmen und ob
3. drittes und drittletztes Symbol übereinstimmen und
4. usw. ...

Idee:

- ▶ Schiebe vordere Worthälfte von links in die mittlere Zelle
- ▶ schiebe hintere Worthälfte von rechts in die mittlere Zelle
- ▶ prüfe, ob in der Mitte immer gleiche Symbole ankommen.

Zweites Beispiel: Erkennung einer Palindromsprache

Aufgabe: $A = \{a, b\}$

akzeptiere $L = L_{\text{pal}} = \{v xv^R \mid v \in A^* \wedge x \in A\}$

Problem: Prüfe, ob

1. erstes und letztes Symbol übereinstimmen und ob
2. zweites und vorletztes Symbol übereinstimmen und ob
3. drittes und drittletztes Symbol übereinstimmen und
4. usw. ...

Idee:

- ▶ Schiebe vordere Worthälfte von links in die mittlere Zelle
- ▶ schiebe hintere Worthälfte von rechts in die mittlere Zelle
- ▶ prüfe, ob in der Mitte immer gleiche Symbole ankommen.

aaaaber ...

Verfeinerung

neue Probleme:

- ▶ Woher „weiß“ eine Zelle, in welcher Worthälfte sie liegt?
- ▶ Wo ist die Mitte?

Verfeinerung

neue Probleme:

- ▶ Woher „weiß“ eine Zelle, in welcher Worthälfte sie liegt?
- ▶ Wo ist die Mitte?

Idee für langsamen Algorithmus:

erst mal feststellen, welche Zellen wo liegen ...

Verfeinerung

neue Probleme:

- ▶ Woher „weiß“ eine Zelle, in welcher Worthälfte sie liegt?
- ▶ Wo ist die Mitte?

Idee für schnellen Algorithmus:

Solange eine Zelle nicht weiß, in wo sie liegt, muss sie *alle eventuell* notwendigen Aktionen durchführen.

Verfeinerung

neue Probleme:

- ▶ Woher „weiß“ eine Zelle, in welcher Worthälfte sie liegt?
- ▶ Wo ist die Mitte?

Idee für schnellen Algorithmus:

Solange eine Zelle nicht weiß, in wo sie liegt, muss sie *alle eventuell* notwendigen Aktionen durchführen.

Also:

- ▶ *Alle* Symbole werden nach links verschoben.
- ▶ *Alle* Symbole werden nach rechts verschoben.
- ▶ *Jede* Zelle tut so, als ob sie die mittlere wäre.
- ▶ Man bestimmt die tatsächliche Mitte des Eingabewortes und benutzt das Ergebnis dieser Zelle.

Bestandteile

- ▶ $N = \{-1, 0, 1\}$ und
- ▶ $Q = A \cup \{\sqcup\} \cup Q_l \times Q_r \times Q_v \times Q_{lr}$ mit
 - ▶ $Q_l = \{\langle a, \langle b, \sqcup \rangle\}$
 - ▶ $Q_r = \{a\rangle, b\rangle, \sqcup\}$
 - ▶ $Q_v = \{+, -\}$
 - ▶ $Q_{lr} = \{\rangle, \langle, \langle +, \langle -, \sqcup\}$

Initialisierungsschritt

für alle $x, y, z \in A$:

$\ell(-1)$	$\ell(0)$	$\ell(1)$	$\delta(\ell)$
\sqcup	y	z	$(\langle y, y\rangle, +, \rangle)$
x	y	z	$(\langle y, y\rangle, +, \sqcup)$
x	y	\sqcup	$(\langle y, y\rangle, +, \langle)$
\sqcup	y	\sqcup	$(\langle y, y\rangle, +, \langle +)$

Überföhrungsfunktion (1)

Ist

$$\ell(-1) = \begin{array}{|c|} \hline \langle x_l \rangle \\ \hline x_r \rangle \\ \hline v_l \\ \hline d_l \\ \hline \end{array} \quad \ell(0) = \begin{array}{|c|} \hline \langle y_l \rangle \\ \hline y_r \rangle \\ \hline v_m \\ \hline d_m \\ \hline \end{array} \quad \ell(1) = \begin{array}{|c|} \hline \langle z_l \rangle \\ \hline z_r \rangle \\ \hline v_r \\ \hline d_r \\ \hline \end{array}$$

so sei

$$\delta(\ell) = \begin{array}{|c|} \hline \langle z_l \rangle \\ \hline x_r \rangle \\ \hline v'_m \\ \hline d'_m \\ \hline \end{array}$$

Überföhrungsfunktion (2)

wobei

$$v'_m = \begin{cases} + & \text{falls } v_m = + \wedge z_l = x_r \\ - & \text{sonst} \end{cases}$$

und

$$d'_m = \begin{cases} d_l & \text{falls } d_m = d_r = \perp \\ d_r & \text{falls } d_m = d_l = \perp \\ <v'_m & \text{falls } d_l = > \wedge d_r = < \\ <- & \text{falls } d_m = > \wedge d_r = < \\ d_m & \text{falls } d_m \in \{<+, <-\} \\ \vdots & \text{usw.} \end{cases}$$

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		b	a	b	a	b	a	a		...
...		<b	<a	<b	<a	<b	<a	<a		...
...		b>	a>	b>	a>	b>	a>	a>		...
...		+	+	+	+	+	+	+		...
...		>						<		...
...		<a	<b	<a	<b	<a	<a			...
...			b>	a>	b>	a>	b>	a>		...
...		-	+	+	+	+	-	-		...
...			>				<			...
...		<b	<a	<b	<a	<a				...
...				b>	a>	b>	a>	b>		...
...		-	-	+	+	-	-	-		...
...				>		<				...
...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...

...		<a	<b	<a	<a					...
...					b>	a>	b>	a>		...
...		-	-	-	-	-	-	-		...
...					<-					...
...		<b	<a	<a						...
...						b>	a>	b>		...
...		-	-	-	-	-	-	-		...
...				<-	<-	<-				...
...		<a	<a							...
...							b>	a>		...
...		-	-	-	-	-	-	-		...
...			<-	<-	<-	<-	<-			...
...		<a								...
...								b>		...
...		-	-	-	-	-	-	-		...
...		<-	<-	<-	<-	<-	<-	<-		...

zur Verifikation (1)

Es sei $|w| = n$, $1 \leq i \leq n$, $0 \leq t \leq n$ und $c^t := \Delta^t(c_w)$.

- ▶ Für die „ersten Register“ $c_i^t[1]$ gilt:
Für $i \leq n + 1 - t$ ist $c_i^t[1] = w_{i+(t-1)}$
- ▶ Für die „zweiten Register“ $c_i^t[2]$ gilt:
Für $i \geq t$ ist $c_i^t[2] = w_{i-(t-1)}$
- ▶ Für die „dritten Register“ $c_i^t[3]$ gilt:
Für $t \leq i \leq n + 1 - t$ ist
 $c_i^t[3] = + \iff w_{i-(t-1)} \cdots w_i \cdots w_{i+(t-1)} \in L_{\text{pal}}$

zur Verifikation (2)

- Für die „vierten Register“ $c_i^t[4]$ gilt z. B., falls n ungerade ist:

$$t < \frac{n+1}{2} \implies c_i^t[4] = \begin{cases} > & \text{falls } i = t \\ < & \text{falls } i = n+1-t \\ \perp & \text{sonst} \end{cases}$$

$$t \geq \frac{n+1}{2} \implies c_i^t[4] = \begin{cases} <+ & \text{falls } w \in L_{\text{pal}} \text{ und } i = n+1-t \\ <- & \text{falls } w \notin L_{\text{pal}} \text{ und } i = n+1-t \\ \perp & \text{sonst} \end{cases}$$

- also insbesondere

$$c_1^n[4] = \begin{cases} <+ & \text{falls } w \in L_{\text{pal}} \\ <- & \text{falls } w \notin L_{\text{pal}} \end{cases}$$

Aufgaben

Konstruiere (möglichst präzise) ZA, die die folgenden formalen Sprachen akzeptieren:

- ▶ $L = \{vcv \mid v \in \{a, b\}^+\} \subset \{a, b, c\}^+$
- ▶ $L = \{vv \mid v \in \{a, b\}^+\} \subset \{a, b, c\}^+$

Schaffen Sie es in Zeit $t(n) = n$?

Kommen Sie mit Nachbarschaft $\{0, 1\}$ aus?

Beides gleichzeitig?

Aufgabe**

Konstruiere (möglichst präzise) ZA, der möglichst schnell die formale Sprache

$$L = \{uvu \mid u, v \in \{a, b\}^* \wedge |u| \geq 2\} \subset \{a, b\}^+$$

akzeptiert.

Zusammenfassung

- ▶ Ruhezustände, tote Zustände
- ▶ Muster, endliche Konfigurationen
- ▶ gegenseitige Simulierbarkeit von TM und ZA für endliche Konfigurationen
- ▶ Erkennung formaler Sprachen mit ZA